运算符

算术运算符

符号有：+　-　*　/　%

他们都是针对数字进行的运算；如果他们两边有不是数字的数据，就会自动转换为数字；其中取余运算，%，它只针对“整数”，如果不是，会自动截取为整数。

12.3 % 3 相当于　11%3　　12% 3.2 相当于　12 % 3

自增自减运算

a) 常规：对数字进行自加 1 或自减 1

b) 字符串：只能自增，且自增的效果就是　下一个字符

c) 布尔值递增减无效

d) Null 递减无效，递增结果为 1

前自增和后自增的区别（自减同自境）

前自增和后自增效果是一样的，如果要赋值给别的变量，前自增是将自增后的值赋值给新的变量，后自增是将值赋值给变量后，再自增。对自加的变量没有影响。

```php
$v1 = 1;
$v2 = 1;

echo $v1++;
echo "<hr>";
echo ++$v2."<hr>";
$a1 = $v1++;
$a2 = ++$v2;
echo "<hr>";
echo $a1."-------".$a2;
echo "<hr>";
$time1 = microtime(true);
for($i = 1;$i<10000000;++$i){}
$time2 = microtime(true);
for($i = 1;$i<10000000;$i++){}
$time3 = microtime(true);
echo "前加加所用时间".($time2-$time1);
echo "<hr>";
echo "后加加所用时间".($time3-$time2);
```

1

2

2-------3

前加加所用时间0.6630380153656

后加加所用时间0.71604084968567

比较运算符

符号：> >=　< <=　== != === !===

比较运算符

例子	名称	结果
$a == $b	等于	TRUE，如果类型转换后 *$a* 等于 *$b*。
$a === $b	全等	TRUE，如果 *$a* 等于 *$b*，并且它们的类型也相同。
$a != $b	不等	TRUE，如果类型转换后 *$a* 不等于 *$b*。
$a <> $b	不等	TRUE，如果类型转换后 *$a* 不等于 *$b*。
$a !== $b	不全等	TRUE，如果 *$a* 不等于 *$b*，或者它们的类型不同。
$a < $b	小与	TRUE，如果 *$a* 严格小于 *$b*。
$a > $b	大于	TRUE，如果 *$a* 严格大于 *$b*。
$a <= $b	小于等于	TRUE，如果 *$a* 小于或者等于 *$b*。
$a >= $b	大于等于	TRUE，如果 *$a* 大于或者等于 *$b*。

a) 一般的比较：是针对数字进行的大小比较

b) ==和===比较：前者通常叫做模糊相等的比较，后者叫做精确相等的比较（只有数据的类型和数据的值或者内容，都相等，才是全等的）。

c) 不要对浮点数直接进行大小比较

常见不同类型（标量类型）数据之间的比较

a) 有布尔值的数据中，有布尔值，转为布尔值比较

b) 有数字转为数字比较

c) 两边都是纯数字字符串，转为数字比较

d) 否则就按字符串比较。字符串比较的规则是：对两边的字符串，一个一个从前往后取出字符并进行比较，谁先大，结果就是谁大。

逻辑运算符

逻辑运算符

例子	名称	结果
$a and $b	And（逻辑与）	**TRUE**，如果 *$a* 和 *$b* 都为 **TRUE**。
$a or $b	Or（逻辑或）	**TRUE**，如果 *$a* 或 *$b* 任一为 **TRUE**。
$a xor $b	Xor（逻辑异或）	**TRUE**，如果 *$a* 或 *$b* 任一为 **TRUE**，但不同时是。
! $a	Not（逻辑非）	**TRUE**，如果 *$a* 不为 **TRUE**。
$a && $b	And（逻辑与）	**TRUE**，如果 *$a* 和 *$b* 都为 **TRUE**。
$a || $b	Or（逻辑或）	**TRUE**，如果 *$a* 或 *$b* 任一为 **TRUE**。

"与"和"或"有两种不同形式运算符的原因是它们运算的优先级不同（见运算符优先级）。

运算符优先级

结合方向	运算符	附加信息
无	clone new	clone 和 new
左	[	array()
右	++ -- ~ (int) (float) (string) (array) (object) (bool) @	类型和递增／递减
无	instanceof	类型
右	!	逻辑运算符
左	* / %	算术运算符
左	+ - .	算术运算符和字符串运算符
左	<< >>	位运算符
无	== != === !== <>	比较运算符
左	&	位运算符和引用
左	^	位运算符
左	|	位运算符
左	&&	逻辑运算符
左	||	逻辑运算符
左	? :	三元运算符
右	= += -= *= /= .= %= &= |= ^= <<= >>= =>	赋值运算符
左	and	逻辑运算符
左	xor	逻辑运算符
左	or	逻辑运算符
左	,	多处用到

字符串运算符　.

赋值运算符　=

衍生的赋值运算

+=　加等　相当于　$变量名 = $变量名 +值

-= 减等　想当于 $变量名 = $变量名 -值

*=　/= %= .= 同上

条件 三目（三元）运算符

数据值 1？数据值 2 ： 数据值 3

对数据值进行判断，如果为“真”，则该运算符的运算结果就是数据值 2，否则就是数据值 3

$fenshu = 66;

$valuation = $fenshu>=60 ? “及格” : “不及格”; //　　最终$valuation 为及格

位运算符

基础：

a) 位：就是 2 进制数字的每一个“位”，一个整数数字，有 32 个位构成

b) 位运算符是仅仅针对整数进行的运算符

位运算符

例子	名称	结果
$a & $b	And（按位与）	将把 $a 和 $b 中都为 1 的位设为 1。
$a | $b	Or（按位或）	将把 $a 和 $b 中任何一个为 1 的位设为 1。
$a ^ $b	Xor（按位异或）	将把 $a 和 $b 中一个为 1 另一个为 0 的位设为 1。
~ $a	Not（按位取反）	将 $a 中为 0 的位设为 1，反之亦然。
$a << $b	Shift left（左移）	将 $a 中的位向左移动 $b 次（每一次移动都表示"乘以 2"）。
$a >> $b	Shift right（右移）	将 $a 中的位向右移动 $b 次（每一次移动都表示"除以 2"）。

c) 位运算符的基本语法规则

按位与基本规则

1 & 1 ==➔ 1

1 & 0 ==➔ 0

0& 1 ==➔0

0 & 0 ==➔0

按位或基本规则

1 | 1 ==➔ 1

1 | 0 ==➔ 1

0| 1 ==➔1

0| 0 ==➔0

按位非基本规则

~1 ==➔ 0

~0 ==➔ 1

按位异或基本规则

1 ^1 ==➔ 0

1 ^ 0 ==➔ 1

0^1 ==➔1

0^ 0 ==➔0

整数的按位左移

$v1 << $m　　将数字 v1 的二进制数字形式（32 位的）的每一个位上的数字都一次的向左边移动 m 位，并将右边空出来的位置补 0，左边超出的舍去。

整数的按位右移